Chapter 3

SYSTEM ANALYSIS

3.1 Introduction

Mobile agents are widely used in providing distributed services because they are able to run over a distributed computing system and they overcome the shortcomings of client-server (CS) architecture that uses a lot of bandwidth. Mobile agents uses object-oriented technology, networks and distributed computing system.

This chapter focuses on the analysis, instrumentation and methodology used to develop this agent.

3.2 Java Aglets

This agent system was developed using Java Aglets 2.0 (Aglets) which uses Agent Transfer Protocol (ATP) to move over the network. Aglet is a Java-based mobile agent system. Java supports network connection, secure execution, object serialization and it is platform independent, hence, is a good language for agent development.

Implementation of this agent system uses the mobile agent paradigm. Mobile agent was chosen compared to static agent or other paradigms for example the CS paradigm because it can move to the server to get a certain service and can move to the client to provide a certain service. In CS paradigm, a client needs to communicate with a server across a network. Therefore mobile agent architecture reduces network bandwidth

problem. In cases of unreliable network connection, mobile agents can also work off-line and communicate the results when the application is back online. In CS, if a network connection is lost, the client has to restart the query. Therefore, a mobile agent is extremely useful when network bandwidth is low and there is no permanent network connection.

Aglets mobile agent is called an aglet which comes from the word 'agent' and 'applet'. This aglet moves from one host to another over a network. It brings the program code and the objects together with it. An aglet moves computation toward resources. It is autonomous and reactive because it responds to incoming messages.

The aglet model uses a few key abstractions. Figure 3.1 below shows how an aglet is related to its proxy, context and the server. The aglet key abstractions are :

1) Aglet – an autonomous mobile Java object that travels in the network.
2) Proxy - a representative of an aglet that serves as a shield of the aglet to prevent any direct access to the aglets' public method and to hide its real location.
3) Context - an aglet's workplace for maintaining and managing running aglets in a uniform execution environment where the host system is secured against malicious aglets. An aglet gets its resources and data from its context.
4) Message - it is the object which is exchanged between aglets to collaborate and exchange information in a loosely coupled fashion.
5) Future reply - used in asynchronous message sending as a handler to receive a result asynchronously later.
6) Identifier - a unique identifier is assigned to each aglet.

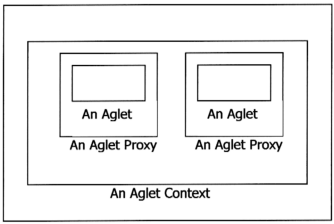

Figure 3.1
The Aglet Environment
(Smith, 1999)

An aglet operates using one of the following five operations:

1) Creation – a new agent is created in a context, given an identifier, initialized and starts execution.
2) Cloning - produces an identical copy of an aglet.
3) Dispatching - moves an aglet from one context to a destination context.
4) Retraction - pulls an aglet from one context to the calling context.
5) Deactivation - temporarily stops an aglet, the aglet releases its resources.
6) Activation - restores an aglet in a context.
7) Disposal - stops an aglet and removes it from the current context.
8) Messaging - exchange of information between aglets. There are two types of messaging: synchronous and asynchronous.
9) Naming - assigns a unique identifier to new aglets.

Aglets comes together with a server called Tahìti that provides an interface to view the running context. Based on Aglets framework, it is possible to define a mobile agent which is able to transfer files on predefined nodes in a network. There are two

important classes when using this Aglets framework, i.e. the *Aglet* class and the *Message* class.

3.2.1 Aglet Class

The *Aglet* class is the most important class in Aglets. The Aglets framework provides a few methods from the *Aglet* class that can be overridden for a customized aglet. Examples of the methods are *onCreation* and *run* methods.

1) *onCreation* method

   This method is for performing specific initialization when the aglet is created.

2) *run* method

   This method is called every time the aglet arrived at or is activated in a new context. To dispatch an aglet to another host, the destination host address or Uniform Resource Locator (URL) is specified in the *run* method using the *dispatch()* command. When *dispatch* is called, the aglet is serialized, preserving the state information of an aglet. When object serialization occurs, an object is converted into a sequential byte representation that is then transferred over the network to a destination host where it is deserialized to recreate the aglet and its state.

3.2.2 Message Class

A *Message* is an object for communication among aglets. A *Message* has two attributes; its kind and its argument. A *Message* is distinguished by its kind. After a *Message* is created, it is handled by the aglet based on its kind.

3.3 Agent Transfer Protocol (ATP)

An aglet uses the Agent Transfer Protocol (ATP) to move in the network. An aglet moves with a *dispatch* command which specifies the host of destination context. When an agent is dispatched, it uses ATP to bring the bytecode and state information to the destination context. Once the agent arrived, the bytecode is deserialized to recreate the agent's state. This can be summarized in Figure 3.2 below.

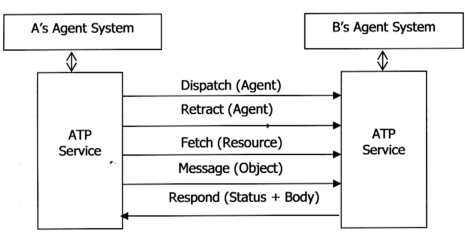

Figure 3.2
Agent Transfer Protocol
(Lange, 1998)

ATP is an application-level protocol for distributed agent-based systems that is platform-independent, enabling it to transfer agents between networked computers. ATP works closely with ATP-based agent service. A machine which hosts agents has

an ATP-based agent service. A machine A wanting to communicate with machine B must connect through a connection between ATP service A and ATP service B. ATP service A and B communicate via request-response paradigm, where A acts as a sender that sends requests to B, the recipient. A then waits for a response from B. This interaction, shown in Figure 3.3 above uses the four request methods of ATP: *dispatch, retract, fetch* and *message*.

1) Dispatch an agent

   Dispatching an agent means dispatching the agent in the request. When the receiver receives the agent, it will send A, the sender, a response in the form of a status code.

2) Retract an agent

   Retracting an agent from agent service B to agent service A is by sending a retract request from A to B. B responds with a status code and the specified agent in its body.

3) Fetch a class

   If agent service B needs to retrieve the agent code from A, B sends a fetch request to A. A responds with a status code and the required agent code.

4) Send a message

   If an agent at A wants to send a message to an agent at B, A sends a messge request and B replies with a status code.

3.4 Security Issues in Agent Implementation

Agent implementation imposes several security issues from agents and hosts perspectives. They are:

1) Authentication of an agent at the destination host. An aglet has its own unique identifier for its lifetime that is used for authentication. To ensure the safety of dispatched aglet, the owner may define its security preferences, for example, who may access the aglet on its itinerary and how to limit the agent's capabilities.

2) The context master is responsible for the safety and security of the context and the machine. Aglets should not interfere with other aglets in the same context unless the aglet owner permits it. The context also needs to protect itself against aglets.

3) An aglet is governed by its manufacturer, owner and the context master. The context master authority is the strongest, followed by the aglet owner and lastly by the aglet manufacturer.

3.5 Analysis Summary

Based on the analysis done in this chapter, this mobile agent system was developed using Java Aglets and operates using Agent Transfer Protocol. This system works on TCP/IP as ATP is an application level protocol for TCP/IP.

Applications that simulate CS paradigm were developed using Java streams and RMI. Network, socket and file system-based streams were used. The mobile agent system and the stream application involve serialization mechanism to enable files to be converted to byte presentation and restored at the destination machine. The detail algorithm is explained in Chapter 4.

3.6 Methodology

The processes carried out in this dissertation are shown in Figure 1.1 in page 7. The figure shows how this system was built from the problem statement in Section 1.2 until the testing phase.

A survey regarding network communication paradigm and mode was first carried out. Four network communication paradigms were analyzed: CS, MA, CoD and ReV. Two different communication modes were also discussed: pull and push mode.

We also performed tools analysis and Aglets 2.0 has been chosen as the development tool with Java as the programming language. UNITAR LAN was chosen as the small-scale network for testing purposes.

The final part is implementation and testing. For implementation, we developed one system using Aglets for MA paradigm, one application using Java streams and another one using RMI. Various sizes of files were transferred using both systems. For each file, testing was done on three pairs of clients and servers. For each pair of clients and servers, we took three readings and an average was then calculated.

We then plotted the average readings of duration versus size of files on a line graph. Based on the graph, we concluded which approach is better between MA and CS.

3.7 Data Collection and Analysis

This agent system was implemented and tested on Universiti Tun Abdul Razak (UNITAR) LAN. A detail of data collection is provided in Chapter 4.